EXHIBIT 258

```
/*
   A C-program for MT19937-64 (2004/9/29 version).
   Coded by Takuji Nishimura and Makoto Matsumoto.

   This is a 64-bit version of Mersenne Twister pseudorandom number
   generator.

   Before using, initialize the state by using init_genrand64(seed)
   or init_by_array64(init_key, key_length).

   Copyright (C) 2004, Makoto Matsumoto and Takuji Nishimura,
   All rights reserved.

   Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions
   are met:

     1. Redistributions of source code must retain the above copyright
        notice, this list of conditions and the following disclaimer.

     2. Redistributions in binary form must reproduce the above copyright
        notice, this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

     3. The names of its contributors may not be used to endorse or promote
        products derived from this software without specific prior written
        permission.

   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
   A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
   CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
   PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
   PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
   LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
   NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
   SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

   References:
   T. Nishimura, ``Tables of 64-bit Mersenne Twisters''
     ACM Transactions on Modeling and
     Computer Simulation 10. (2000) 348--357.
   M. Matsumoto and T. Nishimura,
     ``Mersenne Twister: a 623-dimensionally equidistributed
       uniform pseudorandom number generator''
     ACM Transactions on Modeling and
     Computer Simulation 8. (Jan. 1998) 3--30.

   Any feedback is very welcome.
   http://www.math.hiroshima-u.ac.jp/~m-mat/MT/emt.html
   email: m-mat @ math.sci.hiroshima-u.ac.jp (remove spaces)
*/


#include <stdio.h>

#define NN 312
#define MM 156
#define MATRIX_A 0xB5026F5AA96619E9ULL
#define UM 0xFFFFFFFF80000000ULL /* Most significant 33 bits */
#define LM 0x7FFFFFFFULL /* Least significant 31 bits */
```

```
/* The array for the state vector */
static unsigned long long mt[NN];
/* mti==NN+1 means mt[NN] is not initialized */
static int mti=NN+1;

/* initializes mt[NN] with a seed */
void init_genrand64(unsigned long long seed)
{
    mt[0] = seed;
    for (mti=1; mti<NN; mti++)
        mt[mti] =  (6364136223846793005ULL * (mt[mti-1] ^ (mt[mti-1] >> 62)) + mti);
}

/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
void init_by_array64(unsigned long long init_key[],
                     unsigned long long key_length)
{
    unsigned long long i, j, k;
    init_genrand64(19650218ULL);
    i=1; j=0;
    k = (NN>key_length ? NN : key_length);
    for (; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 62)) * 3935559000370003845ULL))
          + init_key[j] + j; /* non linear */
        i++; j++;
        if (i>=NN) { mt[0] = mt[NN-1]; i=1; }
        if (j>=key_length) j=0;
    }
    for (k=NN-1; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 62)) * 2862933555777941757ULL))
          - i; /* non linear */
        i++;
        if (i>=NN) { mt[0] = mt[NN-1]; i=1; }
    }

    mt[0] = 1ULL << 63; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0, 2^64-1]-interval */
unsigned long long genrand64_int64(void)
{
    int i;
    unsigned long long x;
    static unsigned long long mag01[2]={0ULL, MATRIX_A};

    if (mti >= NN) { /* generate NN words at one time */

        /* if init_genrand64() has not been called, */
        /* a default initial seed is used     */
        if (mti == NN+1)
            init_genrand64(5489ULL);

        for (i=0;i<NN-MM;i++) {
            x = (mt[i]&UM)|(mt[i+1]&LM);
            mt[i] = mt[i+MM] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
        }
        for (;i<NN-1;i++) {
            x = (mt[i]&UM)|(mt[i+1]&LM);
            mt[i] = mt[i+(MM-NN)] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
        }
        x = (mt[NN-1]&UM)|(mt[0]&LM);
        mt[NN-1] = mt[MM-1] ^ (x>>1) ^ mag01[(int)(x&1ULL)];
```

```
        mti = 0;
    }

    x = mt[mti++];

    x ^= (x >> 29) & 0x5555555555555555ULL;
    x ^= (x << 17) & 0x71D67FFFEDA60000ULL;
    x ^= (x << 37) & 0xFFF7EEE000000000ULL;
    x ^= (x >> 43);

    return x;
}

/* generates a random number on [0, 2^63-1]-interval */
long long genrand64_int63(void)
{
    return (long long)(genrand64_int64() >> 1);
}

/* generates a random number on [0,1]-real-interval */
double genrand64_real1(void)
{
    return (genrand64_int64() >> 11) * (1.0/9007199254740991.0);
}

/* generates a random number on [0,1)-real-interval */
double genrand64_real2(void)
{
    return (genrand64_int64() >> 11) * (1.0/9007199254740992.0);
}

/* generates a random number on (0,1)-real-interval */
double genrand64_real3(void)
{
    return ((genrand64_int64() >> 12) + 0.5) * (1.0/4503599627370496.0);
}


int main(void)
{
    int i;
    unsigned long long init[4]={0x12345ULL, 0x23456ULL, 0x34567ULL, 0x45678ULL}, length=4;
    init_by_array64(init, length);
    printf("1000 outputs of genrand64_int64()\n");
    for (i=0; i<1000; i++) {
      printf("%20llu ", genrand64_int64());
      if (i%5==4) printf("\n");
    }
    printf("\n1000 outputs of genrand64_real2()\n");
    for (i=0; i<1000; i++) {
      printf("%10.8f ", genrand64_real2());
      if (i%5==4) printf("\n");
    }
    return 0;
}
```